Name : Izaz Ahamad **Reg.No :** 12318769

Roll : 21 Data : 12.09.2024

a.Load the titanic dataset into a pandas DataFrame

```
import pandas as pd

titanic_df = pd.read_csv(r"P:\data\titanic.csv")
titanic_df
```

```
     PassengerId  Survived  Pclass  \
0              1         0       3
1              2         1       1
2              3         1       3
3              4         1       1
4              5         0       3
..           ...       ...     ...
886          887         0       2
887          888         1       1
888          889         0       3
889          890         1       1
890          891         0       3

                                                  Name     Sex   Age  SibSp
\
0                              Braund, Mr. Owen Harris    male  22.0      1
1    Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1
2                               Heikkinen, Miss. Laina  female  26.0      0
3         Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                             Allen, Mr. William Henry    male  35.0      0
..                                                 ...     ...   ...    ...
886                              Montvila, Rev. Juozas    male  27.0      0
887                       Graham, Miss. Margaret Edith  female  19.0      0
888           Johnston, Miss. Catherine Helen "Carrie"  female   NaN      1
889                              Behr, Mr. Karl Howell    male  26.0      0
890                                Dooley, Mr. Patrick    male  32.0      0

     Parch            Ticket     Fare Cabin Embarked
0        0         A/5 21171   7.2500   NaN        S
1        0          PC 17599  71.2833   C85        C
2        0  STON/O2. 3101282   7.9250   NaN        S
3        0            113803  53.1000  C123        S
4        0            373450   8.0500   NaN        S
..     ...               ...      ...   ...      ...
886      0            211536  13.0000   NaN        S
887      0            112053  30.0000   B42        S
888      2        W./C. 6607  23.4500   NaN        S
```

```
889      0            111369  30.0000  C148        C
890      0            370376   7.7500   NaN        Q

[891 rows x 12 columns]
```

```python
titanic_df = titanic_df.drop(['PassengerId','Name'],axis=1)
titanic_df.head()
```

```
   Survived  Pclass     Sex   Age  SibSp  Parch            Ticket     Fare  \
0         0       3    male  22.0      1      0         A/5 21171   7.2500
1         1       1  female  38.0      1      0          PC 17599  71.2833
2         1       3  female  26.0      0      0  STON/O2. 3101282   7.9250
3         1       1  female  35.0      1      0            113803  53.1000
4         0       3    male  35.0      0      0            373450   8.0500

  Cabin Embarked
0   NaN        S
1   C85        C
2   NaN        S
3  C123        S
4   NaN        S
```

```python
titanic_df['Age'].isnull().sum()
```

```
177
```

```python
titanic_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 10 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Survived  891 non-null    int64
 1   Pclass    891 non-null    int64
 2   Sex       891 non-null    object
 3   Age       714 non-null    float64
 4   SibSp     891 non-null    int64
 5   Parch     891 non-null    int64
 6   Ticket    891 non-null    object
 7   Fare      891 non-null    float64
 8   Cabin     204 non-null    object
 9   Embarked  889 non-null    object
dtypes: float64(2), int64(4), object(4)
memory usage: 69.7+ KB
```

```python
titanic_df['Age'].median()
```

```
28.0
```

b.Handling missing values for the "Age" column

```
titanic_df['Age'] = titanic_df['Age'].fillna(titanic_df['Age'].median())
titanic_df['Age'].isnull().sum()
```

```
0
```

- **We filled the null values with the median of the Age column**

```
from sklearn.preprocessing import OneHotEncoder

onehot = OneHotEncoder()
sex_array = onehot.fit_transform(titanic_df[['Sex']]).toarray()

sex_array

array([[0., 1.],
       [1., 0.],
       [1., 0.],
       ...,
       [1., 0.],
       [0., 1.],
       [0., 1.]])

sex_df =
pd.DataFrame(data=sex_array,columns=onehot.get_feature_names_out(['Sex']))
sex_df.head()

   Sex_female  Sex_male
0         0.0       1.0
1         1.0       0.0
2         1.0       0.0
3         1.0       0.0
4         0.0       1.0
```

c.Convert the sex column into numerical format using one-hot encoding

```
titanic_df = pd.concat([titanic_df,sex_df],axis=1)
titanic_df = titanic_df.drop('Sex',axis=1)
titanic_df.head()

   Survived  Pclass   Age  SibSp  Parch            Ticket     Fare Cabin  \
0         0       3  22.0      1      0         A/5 21171   7.2500   NaN
1         1       1  38.0      1      0          PC 17599  71.2833   C85
2         1       3  26.0      0      0  STON/O2. 3101282   7.9250   NaN
3         1       1  35.0      1      0            113803  53.1000  C123
4         0       3  35.0      0      0            373450   8.0500   NaN

  Embarked  Sex_female  Sex_male
0        S         0.0       1.0
1        C         1.0       0.0
2        S         1.0       0.0
```

```
3         S         1.0       0.0
4         S         0.0       1.0
```

```
titanic_df['Fare'].mean()
```

```
32.204207968574636
```

d.Normalize the "fare" column

```
from sklearn.preprocessing import StandardScaler

normalize = StandardScaler()
titanic_df['Fare'] = normalize.fit_transform(titanic_df[['Fare']])

titanic_df
```

```
     Survived  Pclass   Age  SibSp  Parch            Ticket      Fare Cabin  \
0           0       3  22.0      1      0         A/5 21171 -0.502445   NaN
1           1       1  38.0      1      0          PC 17599  0.786845   C85
2           1       3  26.0      0      0  STON/O2. 3101282 -0.488854   NaN
3           1       1  35.0      1      0            113803  0.420730  C123
4           0       3  35.0      0      0            373450 -0.486337   NaN
..        ...     ...   ...    ...    ...               ...       ...   ...
886         0       2  27.0      0      0            211536 -0.386671   NaN
887         1       1  19.0      0      0            112053 -0.044381   B42
888         0       3  28.0      1      2        W./C. 6607 -0.176263   NaN
889         1       1  26.0      0      0            111369 -0.044381  C148
890         0       3  32.0      0      0            370376 -0.492378   NaN

    Embarked  Sex_female  Sex_male
0          S         0.0       1.0
1          C         1.0       0.0
2          S         1.0       0.0
3          S         1.0       0.0
4          S         0.0       1.0
..       ...         ...       ...
886        S         0.0       1.0
887        S         1.0       0.0
888        S         1.0       0.0
889        C         0.0       1.0
890        Q         0.0       1.0

[891 rows x 11 columns]
```

```
titanic_df['Fare'].mean()
```

```
3.987332972840069e-18
```

d.Spliting the data into training(80%) and testing(20%)

```
X = titanic_df.drop('Survived',axis=1)
Y = titanic_df['Survived']
```

```python
from sklearn.model_selection import train_test_split

x_train,x_test,y_train,y_test =
train_test_split(X,Y,test_size=0.2,random_state=42)
print(x_train.shape,y_train.shape,x_test.shape,y_test.shape)
```

```
(712, 10) (712,) (179, 10) (179,)
```

```python
titanic_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 11 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Survived    891 non-null    int64
 1   Pclass      891 non-null    int64
 2   Age         891 non-null    float64
 3   SibSp       891 non-null    int64
 4   Parch       891 non-null    int64
 5   Ticket      891 non-null    object
 6   Fare        891 non-null    float64
 7   Cabin       204 non-null    object
 8   Embarked    889 non-null    object
 9   Sex_female  891 non-null    float64
 10  Sex_male    891 non-null    float64
dtypes: float64(4), int64(4), object(3)
memory usage: 76.7+ KB
```

cabin column has high missing values, bettter to drop the column

```python
titanic_df = titanic_df.drop('Cabin',axis=1)
titanic_df
```

```
     Survived  Pclass   Age  SibSp  Parch            Ticket      Fare  \
0           0       3  22.0      1      0         A/5 21171 -0.502445
1           1       1  38.0      1      0          PC 17599  0.786845
2           1       3  26.0      0      0  STON/O2. 3101282 -0.488854
3           1       1  35.0      1      0            113803  0.420730
4           0       3  35.0      0      0            373450 -0.486337
..        ...     ...   ...    ...    ...               ...       ...
886         0       2  27.0      0      0            211536 -0.386671
887         1       1  19.0      0      0            112053 -0.044381
888         0       3  28.0      1      2        W./C. 6607 -0.176263
889         1       1  26.0      0      0            111369 -0.044381
890         0       3  32.0      0      0            370376 -0.492378

    Embarked  Sex_female  Sex_male
0          S         0.0       1.0
1          C         1.0       0.0
2          S         1.0       0.0
3          S         1.0       0.0
```

```
4          S         0.0       1.0
..       ...         ...       ...
886        S         0.0       1.0
887        S         1.0       0.0
888        S         1.0       0.0
889        C         0.0       1.0
890        Q         0.0       1.0

[891 rows x 10 columns]
```

Question - 2

```
onehot1 = OneHotEncoder()
emb_array = onehot1.fit_transform(titanic_df[['Embarked']]).toarray()
emb_df =
pd.DataFrame(data=emb_array,columns=onehot1.get_feature_names_out(['Embarked'
]))
titanic_df = pd.concat([titanic_df,emb_df],axis=1)
titanic_df = titanic_df.drop('Embarked',axis=1)


X = titanic_df.drop('Survived',axis=1)
Y = titanic_df['Survived']

x_train,x_test,y_train,y_test =
train_test_split(X,Y,test_size=0.2,random_state=42)
print(x_train.shape,y_train.shape,x_test.shape,y_test.shape)

(712, 9) (712,) (179, 9) (179,)

scaler1 = StandardScaler()
x_train_scaled = scaler1.fit_transform(x_train)
x_train_scaled

array([[-1.61413602,  1.25364106, -0.47072241, ..., -0.46146201,
        -0.30335547,  0.59681695],
       [-0.40055118, -0.47728355, -0.47072241, ..., -0.46146201,
        -0.30335547,  0.59681695],
       [ 0.81303367,  0.21508629, -0.47072241, ..., -0.46146201,
        -0.30335547,  0.59681695],
       ...,
       [ 0.81303367,  0.90745614,  1.23056874, ..., -0.46146201,
        -0.30335547,  0.59681695],
       [-1.61413602, -1.1696534 ,  0.37992316, ..., -0.46146201,
        -0.30335547,  0.59681695],
       [-1.61413602, -0.63114352, -0.47072241, ..., -0.46146201,
        -0.30335547,  0.59681695]])

scaler2 = StandardScaler()
x_test_scaled = scaler1.fit_transform(x_test)
x_test_scaled
```

```
array([[ 0.88742288, -0.15235845,  0.82036305, ...,  1.77842366,
        -0.32394177, -1.40830868],
       [-0.25537349,  0.07757294, -0.55202   , ..., -0.56229571,
        -0.32394177,  0.7100716 ],
       [ 0.88742288, -0.76550883, -0.55202   , ..., -0.56229571,
        -0.32394177,  0.7100716 ],
       ...,
       [ 0.88742288,  0.61407953,  0.82036305, ..., -0.56229571,
        -0.32394177,  0.7100716 ],
       [-0.25537349, -0.99544022, -0.55202   , ..., -0.56229571,
        -0.32394177,  0.7100716 ],
       [ 0.88742288, -1.99180959,  0.82036305, ..., -0.56229571,
        -0.32394177,  0.7100716 ]])
```

```
titanic_df.shape #(891, 10)
```

```
(891, 10)
```

2.Question

Build a Sequential neural network using TensorFlow/Keras for binary classification with the following structure:

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import Sequential,layers
from tensorflow.keras.layers import Dense
```

- Input layer with the appropriate number of features.
- One hidden layer with 16 neurons and ReLU activation.
- Another hidden layer with 8 neurons and ReLU activation.
- Output layer with 1 neuron and sigmoid activation.

```
model = Sequential()
#Input layer with the appropriate number of features
model.add(Dense(16,activation='relu',input_dim=9))

#One hidden layer with 16 neurons and ReLU activation
model.add(Dense(8,activation='relu'))

#Output layer with 1 neuron and sigmoid activation
model.add(Dense(1,activation='sigmoid'))
model.summary()
```

```
Model: "sequential_5"
```

Layer (type)	Output Shape	Param #
dense_16 (Dense)	(None, 16)	160

dense_17 (Dense)	(None, 8)	136
dense_18 (Dense)	(None, 1)	9

```
Total params: 305 (1.19 KB)

Trainable params: 305 (1.19 KB)

Non-trainable params: 0 (0.00 B)
```

b. Compile the model usingappropriate loss function and optimizer.

```
model.compile(optimizer='Adam',loss='binary_crossentropy',metrics=['accuracy'
])

model.fit(x_train_scaled,y_train,epochs=50,batch_size=15,verbose=2)

Epoch 1/50
48/48 - 0s - 2ms/step - accuracy: 0.8469 - loss: 0.3588
Epoch 2/50
48/48 - 0s - 2ms/step - accuracy: 0.8469 - loss: 0.3590
Epoch 3/50
48/48 - 0s - 2ms/step - accuracy: 0.8469 - loss: 0.3595
Epoch 4/50
48/48 - 0s - 2ms/step - accuracy: 0.8469 - loss: 0.3582
Epoch 5/50
48/48 - 0s - 2ms/step - accuracy: 0.8399 - loss: 0.3592
Epoch 6/50
48/48 - 0s - 2ms/step - accuracy: 0.8469 - loss: 0.3582
Epoch 7/50
48/48 - 0s - 2ms/step - accuracy: 0.8483 - loss: 0.3579
Epoch 8/50
48/48 - 0s - 2ms/step - accuracy: 0.8469 - loss: 0.3582
Epoch 9/50
48/48 - 0s - 2ms/step - accuracy: 0.8511 - loss: 0.3573
Epoch 10/50
48/48 - 0s - 2ms/step - accuracy: 0.8469 - loss: 0.3569
Epoch 11/50
48/48 - 0s - 2ms/step - accuracy: 0.8469 - loss: 0.3569
Epoch 12/50
48/48 - 0s - 2ms/step - accuracy: 0.8483 - loss: 0.3560
Epoch 13/50
48/48 - 0s - 2ms/step - accuracy: 0.8483 - loss: 0.3567
Epoch 14/50
48/48 - 0s - 2ms/step - accuracy: 0.8497 - loss: 0.3565
Epoch 15/50
48/48 - 0s - 2ms/step - accuracy: 0.8427 - loss: 0.3553
Epoch 16/50
48/48 - 0s - 2ms/step - accuracy: 0.8483 - loss: 0.3552
Epoch 17/50
```

```
48/48 - 0s - 4ms/step - accuracy: 0.8455 - loss: 0.3568
Epoch 18/50
48/48 - 0s - 3ms/step - accuracy: 0.8469 - loss: 0.3557
Epoch 19/50
48/48 - 0s - 2ms/step - accuracy: 0.8497 - loss: 0.3539
Epoch 20/50
48/48 - 0s - 2ms/step - accuracy: 0.8497 - loss: 0.3539
Epoch 21/50
48/48 - 0s - 2ms/step - accuracy: 0.8497 - loss: 0.3540
Epoch 22/50
48/48 - 0s - 2ms/step - accuracy: 0.8483 - loss: 0.3542
Epoch 23/50
48/48 - 0s - 2ms/step - accuracy: 0.8483 - loss: 0.3536
Epoch 24/50
48/48 - 0s - 2ms/step - accuracy: 0.8483 - loss: 0.3526
Epoch 25/50
48/48 - 0s - 2ms/step - accuracy: 0.8483 - loss: 0.3540
Epoch 26/50
48/48 - 0s - 2ms/step - accuracy: 0.8469 - loss: 0.3528
Epoch 27/50
48/48 - 0s - 2ms/step - accuracy: 0.8483 - loss: 0.3525
Epoch 28/50
48/48 - 0s - 3ms/step - accuracy: 0.8483 - loss: 0.3528
Epoch 29/50
48/48 - 0s - 3ms/step - accuracy: 0.8469 - loss: 0.3524
Epoch 30/50
48/48 - 0s - 2ms/step - accuracy: 0.8497 - loss: 0.3512
Epoch 31/50
48/48 - 0s - 2ms/step - accuracy: 0.8511 - loss: 0.3515
Epoch 32/50
48/48 - 0s - 2ms/step - accuracy: 0.8455 - loss: 0.3515
Epoch 33/50
48/48 - 0s - 2ms/step - accuracy: 0.8497 - loss: 0.3529
Epoch 34/50
48/48 - 0s - 3ms/step - accuracy: 0.8511 - loss: 0.3511
Epoch 35/50
48/48 - 0s - 3ms/step - accuracy: 0.8497 - loss: 0.3507
Epoch 36/50
48/48 - 0s - 3ms/step - accuracy: 0.8539 - loss: 0.3521
Epoch 37/50
48/48 - 0s - 3ms/step - accuracy: 0.8525 - loss: 0.3507
Epoch 38/50
48/48 - 0s - 3ms/step - accuracy: 0.8497 - loss: 0.3507
Epoch 39/50
48/48 - 0s - 3ms/step - accuracy: 0.8455 - loss: 0.3511
Epoch 40/50
48/48 - 0s - 2ms/step - accuracy: 0.8483 - loss: 0.3504
Epoch 41/50
48/48 - 0s - 3ms/step - accuracy: 0.8497 - loss: 0.3496
Epoch 42/50
```

```
48/48 - 0s - 3ms/step - accuracy: 0.8497 - loss: 0.3480
Epoch 43/50
48/48 - 0s - 3ms/step - accuracy: 0.8469 - loss: 0.3484
Epoch 44/50
48/48 - 0s - 3ms/step - accuracy: 0.8483 - loss: 0.3490
Epoch 45/50
48/48 - 0s - 3ms/step - accuracy: 0.8483 - loss: 0.3478
Epoch 46/50
48/48 - 0s - 3ms/step - accuracy: 0.8497 - loss: 0.3490
Epoch 47/50
48/48 - 0s - 3ms/step - accuracy: 0.8511 - loss: 0.3482
Epoch 48/50
48/48 - 0s - 3ms/step - accuracy: 0.8525 - loss: 0.3463
Epoch 49/50
48/48 - 0s - 3ms/step - accuracy: 0.8511 - loss: 0.3478
Epoch 50/50
48/48 - 0s - 2ms/step - accuracy: 0.8525 - loss: 0.3492

<keras.src.callbacks.history.History at 0x1db8621cad0>
```

c. Train the model for different epochs and report the accuracy on the test set. (mention the epoch on which you have accuracy greater that 85%)

- Epoch 31/50
- 48/48 - 0s - 2ms/step - accuracy: 0.8511 - loss: 0.3515
- 85% on 31st epoch

3 - Question

a. Evaluate the model on the test data and report the test accuracy

```
test_loss, test_accuracy = model.evaluate(x_test_scaled, y_test)
print(f"Test Accuracy: {test_accuracy * 100:.2f}%")
```

```
6/6 ━━━━━━━━━━━━━━━━━━━━━0s 2ms/step - accuracy: 0.8053 - loss: 0.4534
Test Accuracy: 79.89%
```

```
history =
model.fit(x_train_scaled,y_train,epochs=50,batch_size=15,validation_data=(x_t
est_scaled, y_test))
```

```
Epoch 1/50
48/48 ━━━━━━━━━━━━━━━━━━━━━0s 4ms/step - accuracy: 0.8294 - loss: 0.3527 -
val_accuracy: 0.7989 - val_loss: 0.4661
Epoch 2/50
48/48 ━━━━━━━━━━━━━━━━━━━━━0s 3ms/step - accuracy: 0.8561 - loss: 0.3438 -
val_accuracy: 0.8045 - val_loss: 0.4661
```

```
Epoch 3/50
48/48 ━━━━━━━━━━━━━━━━━━━━0s 3ms/step - accuracy: 0.8569 - loss: 0.3350 -
val_accuracy: 0.8045 - val_loss: 0.4653
Epoch 4/50
48/48 ━━━━━━━━━━━━━━━━━━━━0s 3ms/step - accuracy: 0.8416 - loss: 0.3498 -
val_accuracy: 0.7989 - val_loss: 0.4630
Epoch 5/50
48/48 ━━━━━━━━━━━━━━━━━━━━0s 3ms/step - accuracy: 0.8593 - loss: 0.3344 -
val_accuracy: 0.8045 - val_loss: 0.4699
Epoch 6/50
48/48 ━━━━━━━━━━━━━━━━━━━━0s 3ms/step - accuracy: 0.8403 - loss: 0.3590 -
val_accuracy: 0.7989 - val_loss: 0.4736
Epoch 7/50
48/48 ━━━━━━━━━━━━━━━━━━━━0s 3ms/step - accuracy: 0.8250 - loss: 0.3785 -
val_accuracy: 0.7933 - val_loss: 0.4740
Epoch 8/50
48/48 ━━━━━━━━━━━━━━━━━━━━0s 3ms/step - accuracy: 0.8311 - loss: 0.3688 -
val_accuracy: 0.7989 - val_loss: 0.4636
Epoch 9/50
48/48 ━━━━━━━━━━━━━━━━━━━━0s 3ms/step - accuracy: 0.8221 - loss: 0.3830 -
val_accuracy: 0.7933 - val_loss: 0.4741
Epoch 10/50
48/48 ━━━━━━━━━━━━━━━━━━━━0s 4ms/step - accuracy: 0.8362 - loss: 0.3676 -
val_accuracy: 0.7933 - val_loss: 0.4679
Epoch 11/50
48/48 ━━━━━━━━━━━━━━━━━━━━0s 3ms/step - accuracy: 0.8793 - loss: 0.3141 -
val_accuracy: 0.7933 - val_loss: 0.4676
Epoch 12/50
48/48 ━━━━━━━━━━━━━━━━━━━━0s 3ms/step - accuracy: 0.8716 - loss: 0.3204 -
val_accuracy: 0.7933 - val_loss: 0.4678
Epoch 13/50
48/48 ━━━━━━━━━━━━━━━━━━━━0s 3ms/step - accuracy: 0.8481 - loss: 0.3337 -
val_accuracy: 0.7989 - val_loss: 0.4678
Epoch 14/50
48/48 ━━━━━━━━━━━━━━━━━━━━0s 3ms/step - accuracy: 0.8451 - loss: 0.3454 -
val_accuracy: 0.7989 - val_loss: 0.4697
Epoch 15/50
48/48 ━━━━━━━━━━━━━━━━━━━━0s 3ms/step - accuracy: 0.8704 - loss: 0.3167 -
val_accuracy: 0.7933 - val_loss: 0.4697
Epoch 16/50
48/48 ━━━━━━━━━━━━━━━━━━━━0s 3ms/step - accuracy: 0.8530 - loss: 0.3344 -
val_accuracy: 0.7989 - val_loss: 0.4699
Epoch 17/50
48/48 ━━━━━━━━━━━━━━━━━━━━0s 3ms/step - accuracy: 0.8652 - loss: 0.3409 -
val_accuracy: 0.7933 - val_loss: 0.4682
Epoch 18/50
48/48 ━━━━━━━━━━━━━━━━━━━━0s 2ms/step - accuracy: 0.8464 - loss: 0.3594 -
val_accuracy: 0.7933 - val_loss: 0.4748
Epoch 19/50
48/48 ━━━━━━━━━━━━━━━━━━━━0s 2ms/step - accuracy: 0.8711 - loss: 0.3150 -
```

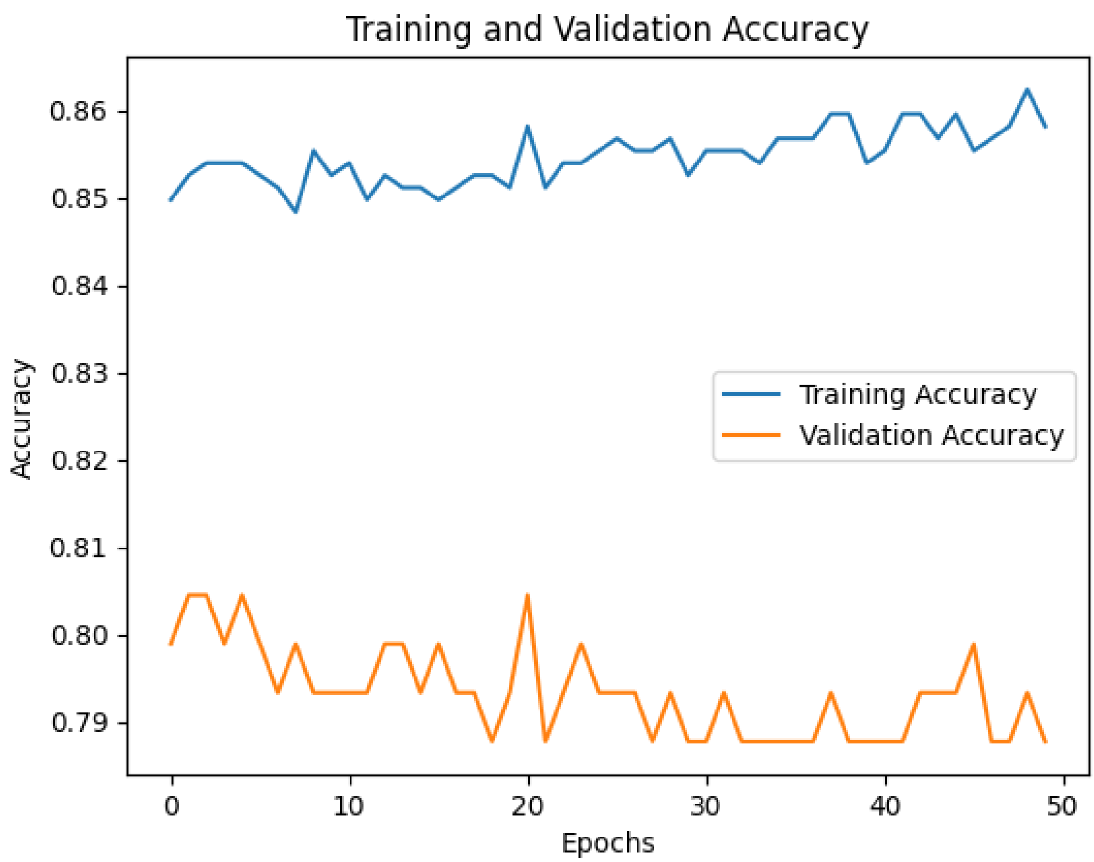
```
val_accuracy: 0.7877 - val_loss: 0.4708
Epoch 20/50
48/48 ━━━━━━━━━━━━━━━━━━━━0s 3ms/step - accuracy: 0.8408 - loss: 0.3508 -
val_accuracy: 0.7933 - val_loss: 0.4727
Epoch 21/50
48/48 ━━━━━━━━━━━━━━━━━━━━0s 3ms/step - accuracy: 0.8599 - loss: 0.3565 -
val_accuracy: 0.8045 - val_loss: 0.4759
Epoch 22/50
48/48 ━━━━━━━━━━━━━━━━━━━━0s 4ms/step - accuracy: 0.8566 - loss: 0.3300 -
val_accuracy: 0.7877 - val_loss: 0.4717
Epoch 23/50
48/48 ━━━━━━━━━━━━━━━━━━━━0s 3ms/step - accuracy: 0.8403 - loss: 0.3653 -
val_accuracy: 0.7933 - val_loss: 0.4754
Epoch 24/50
48/48 ━━━━━━━━━━━━━━━━━━━━0s 4ms/step - accuracy: 0.8429 - loss: 0.3635 -
val_accuracy: 0.7989 - val_loss: 0.4781
Epoch 25/50
48/48 ━━━━━━━━━━━━━━━━━━━━0s 3ms/step - accuracy: 0.8706 - loss: 0.3165 -
val_accuracy: 0.7933 - val_loss: 0.4725
Epoch 26/50
48/48 ━━━━━━━━━━━━━━━━━━━━0s 3ms/step - accuracy: 0.8489 - loss: 0.3670 -
val_accuracy: 0.7933 - val_loss: 0.4715
Epoch 27/50
48/48 ━━━━━━━━━━━━━━━━━━━━0s 3ms/step - accuracy: 0.8591 - loss: 0.3356 -
val_accuracy: 0.7933 - val_loss: 0.4694
Epoch 28/50
48/48 ━━━━━━━━━━━━━━━━━━━━0s 3ms/step - accuracy: 0.8537 - loss: 0.3390 -
val_accuracy: 0.7877 - val_loss: 0.4714
Epoch 29/50
48/48 ━━━━━━━━━━━━━━━━━━━━0s 3ms/step - accuracy: 0.8581 - loss: 0.3278 -
val_accuracy: 0.7933 - val_loss: 0.4749
Epoch 30/50
48/48 ━━━━━━━━━━━━━━━━━━━━0s 3ms/step - accuracy: 0.8707 - loss: 0.3335 -
val_accuracy: 0.7877 - val_loss: 0.4723
Epoch 31/50
48/48 ━━━━━━━━━━━━━━━━━━━━0s 3ms/step - accuracy: 0.8655 - loss: 0.3231 -
val_accuracy: 0.7877 - val_loss: 0.4775
Epoch 32/50
48/48 ━━━━━━━━━━━━━━━━━━━━0s 3ms/step - accuracy: 0.8469 - loss: 0.3482 -
val_accuracy: 0.7933 - val_loss: 0.4813
Epoch 33/50
48/48 ━━━━━━━━━━━━━━━━━━━━0s 4ms/step - accuracy: 0.8548 - loss: 0.3445 -
val_accuracy: 0.7877 - val_loss: 0.4788
Epoch 34/50
48/48 ━━━━━━━━━━━━━━━━━━━━0s 3ms/step - accuracy: 0.8599 - loss: 0.3313 -
val_accuracy: 0.7877 - val_loss: 0.4769
Epoch 35/50
48/48 ━━━━━━━━━━━━━━━━━━━━0s 4ms/step - accuracy: 0.8753 - loss: 0.3244 -
val_accuracy: 0.7877 - val_loss: 0.4727
Epoch 36/50
```

```
48/48 ━━━━━━━━━━━━━━━━━━━━0s 3ms/step - accuracy: 0.8613 - loss: 0.3402 -
val_accuracy: 0.7877 - val_loss: 0.4756
Epoch 37/50
48/48 ━━━━━━━━━━━━━━━━━━━━0s 3ms/step - accuracy: 0.8836 - loss: 0.2953 -
val_accuracy: 0.7877 - val_loss: 0.4836
Epoch 38/50
48/48 ━━━━━━━━━━━━━━━━━━━━0s 3ms/step - accuracy: 0.8655 - loss: 0.3369 -
val_accuracy: 0.7933 - val_loss: 0.4764
Epoch 39/50
48/48 ━━━━━━━━━━━━━━━━━━━━0s 4ms/step - accuracy: 0.8625 - loss: 0.3253 -
val_accuracy: 0.7877 - val_loss: 0.4794
Epoch 40/50
48/48 ━━━━━━━━━━━━━━━━━━━━0s 3ms/step - accuracy: 0.8586 - loss: 0.3400 -
val_accuracy: 0.7877 - val_loss: 0.4832
Epoch 41/50
48/48 ━━━━━━━━━━━━━━━━━━━━0s 3ms/step - accuracy: 0.8413 - loss: 0.3503 -
val_accuracy: 0.7877 - val_loss: 0.4804
Epoch 42/50
48/48 ━━━━━━━━━━━━━━━━━━━━0s 3ms/step - accuracy: 0.8578 - loss: 0.3622 -
val_accuracy: 0.7877 - val_loss: 0.4760
Epoch 43/50
48/48 ━━━━━━━━━━━━━━━━━━━━0s 3ms/step - accuracy: 0.8470 - loss: 0.3559 -
val_accuracy: 0.7933 - val_loss: 0.4811
Epoch 44/50
48/48 ━━━━━━━━━━━━━━━━━━━━0s 3ms/step - accuracy: 0.8467 - loss: 0.3269 -
val_accuracy: 0.7933 - val_loss: 0.4803
Epoch 45/50
48/48 ━━━━━━━━━━━━━━━━━━━━0s 4ms/step - accuracy: 0.8521 - loss: 0.3334 -
val_accuracy: 0.7933 - val_loss: 0.4817
Epoch 46/50
48/48 ━━━━━━━━━━━━━━━━━━━━0s 3ms/step - accuracy: 0.8500 - loss: 0.3241 -
val_accuracy: 0.7989 - val_loss: 0.4818
Epoch 47/50
48/48 ━━━━━━━━━━━━━━━━━━━━0s 3ms/step - accuracy: 0.8704 - loss: 0.3205 -
val_accuracy: 0.7877 - val_loss: 0.4801
Epoch 48/50
48/48 ━━━━━━━━━━━━━━━━━━━━0s 4ms/step - accuracy: 0.8406 - loss: 0.3602 -
val_accuracy: 0.7877 - val_loss: 0.4777
Epoch 49/50
48/48 ━━━━━━━━━━━━━━━━━━━━0s 3ms/step - accuracy: 0.8662 - loss: 0.3307 -
val_accuracy: 0.7933 - val_loss: 0.4838
Epoch 50/50
48/48 ━━━━━━━━━━━━━━━━━━━━0s 3ms/step - accuracy: 0.8494 - loss: 0.3453 -
val_accuracy: 0.7877 - val_loss: 0.4809
```

b. Plot the training and validation accuracy curves over the epochs

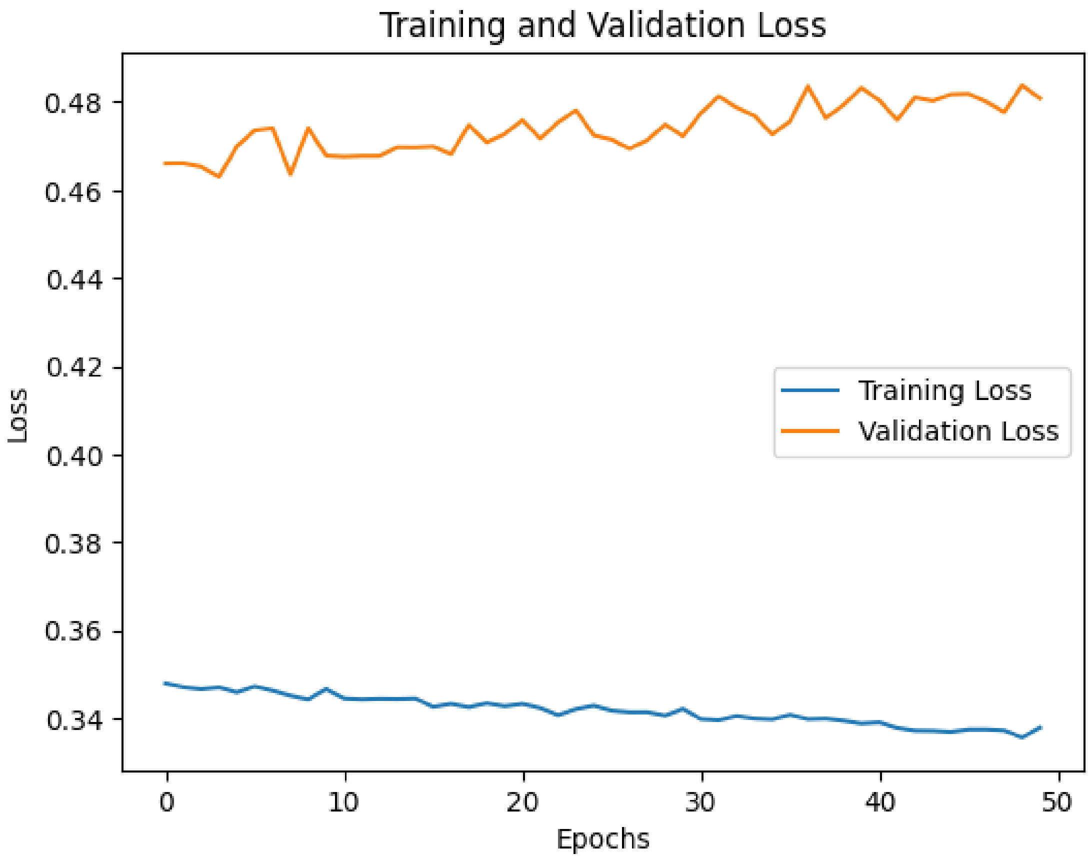
```python
import matplotlib.pyplot as plt

# Plot accuracy curves
```

```
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



c. Plot the training and validation loss curves over the epochs

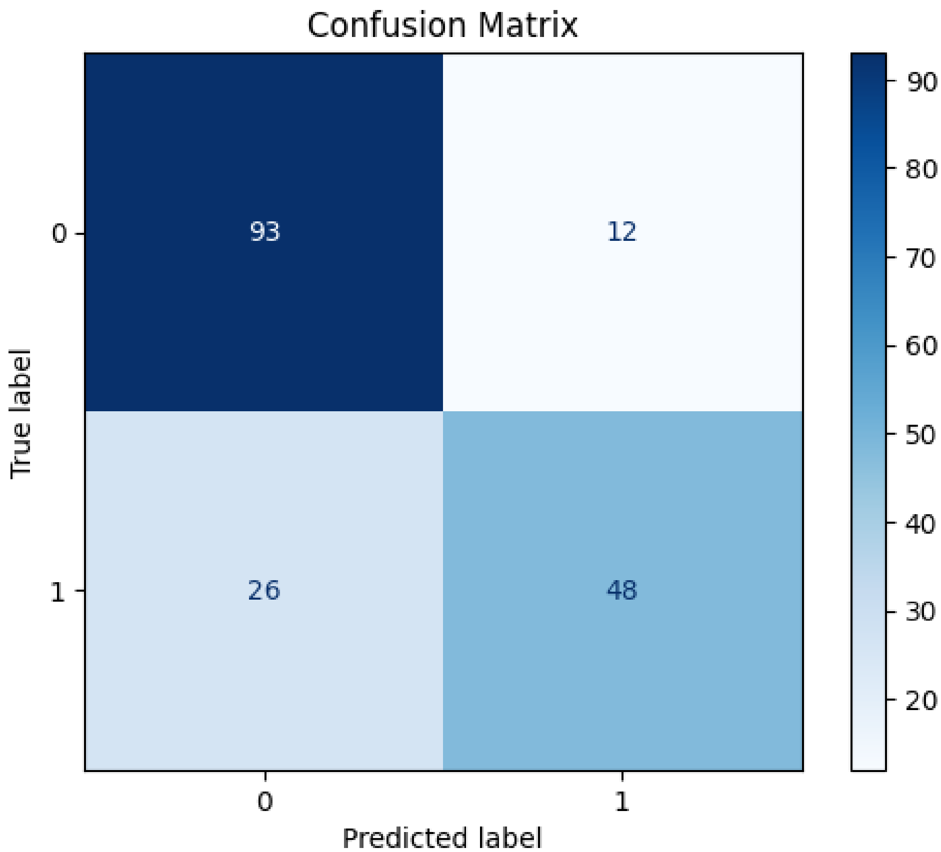
```
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



d. Confusion matrix and explanation (optional)

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Generate predictions (rounded for binary classification)
y_pred = model.predict(x_test_scaled)
y_pred_rounded = (y_pred > 0.5).astype(int)

# Compute confusion matrix
cm = confusion_matrix(y_test, y_pred_rounded)

# Plot the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.show()
```

```
6/6 ━━━━━━━━━━━━━━━━━━━━0s 9ms/step
```


Confusion Matrix
93
12
26
48
0
1
True label
0
1
Predicted label
90
80
70
60
50
40
30
20